Part C

Techniques Used

1. Multithreading Network Connections
2. Sorting (Recursive Quicksort)
3. Implementing Parcelable Interface
4. Image Serialization
5. PHP Script to Make SQL Queries
6. JSON & Gson Java Library
7. Writing to PDF File

1. Multithreading Network Connections

Since network operations are time consuming, yet essential to accessing user data from the online database, I decided to run them on a separate thread from the GUI, so that the GUI remains responsive throughout. This was achieved through inheritance, with each Connection Class inheriting AsyncTask[1], shown in *Figure 1*.

```
static class LoginConnection extends AsyncTask<String, String, String> {

    private Context context;
    public void setContext(Context c) { context = c; }
    ProgressDialog dialog;

    @Override
    protected String doInBackground(String... params) {
        // Do the long network operation here on a separate thread.
        String text;

        try {
            // Make the POST request to the server.
            URL url = new URL("http://trackmycas.com/login.php");
            URLConnection conn = url.openConnection();
            conn.setDoOutput(true);
            OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
            wr.write(params[0]);
            wr.flush();

            // Get the response from the server.
            BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
            StringBuilder sb = new StringBuilder();
            String line = null;

            // Read Server Response
            while((line = br.readLine()) != null)
            {
                // Append server response in string
                sb.append(line);
            }

            text = sb.toString();
            wr.close();
            br.close();
        } catch (Exception e) {
            Log.d("Connection", "Could not connect to the server.");
            e.printStackTrace();
            text = "Could not connect to the server.";
        }
        return text;
    }
```

[1] https://developer.android.com/reference/android/os/AsyncTask.html

Figure 1

All of the code in the overridden method, doInBackground is run on a different thread, and this **allows for future extensibility** to be undertaken easily, accessing the separate thread by modifying only one method. Using this technique for multithreading **shows ingenuity** since overriding the onPreExecute and onPostExecute methods from AsyncTask makes displaying a loading dialog and handling the network result easy to achieve, as seen in *Figure 2*.

```
@Override
protected void onPostExecute(String result) {
    // Handle the result of the network operation.
    dialog.dismiss();
    Log.d("LOGIN_RESPONSE", result);

    // If the login details are correct:
    if (!result.equalsIgnoreCase("-1")) {
        Log.d("OUTP", result);
        String[] userDetails = result.split(",");

        // Store the users details in SharedPreferences for faster login next time.
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
        SharedPreferences.Editor editor = prefs.edit();
        editor.putString("email",userDetails[2]);
        editor.putString("name",userDetails[0]+ " " + userDetails[1]);
        editor.apply();

        // Send the user to the Home Activity.
        Intent intent = new Intent(context, HomeActivity.class);
        LoginActivity.user.setFirstName(userDetails[0]);
        LoginActivity.user.setLastName(userDetails[1]);
        LoginActivity.user.setUserId(Long.parseLong(userDetails[3]));
        intent.putExtra("user", LoginActivity.user);
        context.startActivity(intent);
        ((Activity) context).finish();

        Toast.makeText(context, "Welcome back, " + prefs.getString("name", ""), Toast.LENGTH_LONG).show();
    } else {
        // If the login details are wrong, tell the user.
        Toast.makeText(context, "Could not log in. Email or Password are incorrect.", Toast.LENGTH_SHORT).show();
    }
}

@Override
protected void onPreExecute() {
    // Display loading dialog before network connection.
    dialog = ProgressDialog.show(context, "", "Logging in...", true);
}
```

Figure 2

2. Sorting (Recursive Quicksort)

One essential part of my project is that the experiences in the 'Near Me' section are sorted in order of closest to furthest from the user. A quicksort algorithm[2] was an **ingenious solution** because it has an average efficiency of O(n log n)[3], which is preferable to the O(n^2) efficiency seen in selection sort.[4]

```
int partition(LocationExperience[] arr, int lowerBound, int upperBound)
{
    int i = lowerBound, j = upperBound;
    LocationExperience tmp;
    double pivot = arr[(lowerBound + upperBound) / 2].getDist();

    while (i <= j) {
        while (arr[i].getDist() < pivot)
            i++;
        while (arr[j].getDist() > pivot)
            j--;
        if (i <= j) {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i++;
            j--;
        }
    }

    return i;
}

void quickSort(LocationExperience[] arr, int lowerBound, int upperBound) {
    int index = partition(arr, lowerBound, upperBound);
    if (lowerBound < index - 1)
        quickSort(arr, lowerBound, index - 1);
    if (index < upperBound)
        quickSort(arr, index, upperBound);
}
```

Figure 3

Figure 3 shows how the quicksort algorithm has been implemented. The quicksort method is called and passed an array of LocationExperience objects, which need to be ordered by their distance fields. The lower bound is passed as 0 for the first call and the upper bound is passed as one less than the length of the array. Since quicksort is recursive, it keeps getting called with varying parameters until the array is sorted. This solution **is clever** as it sorts objects based on a field from the object.

---

[2] http://www.java2novice.com/java-sorting-algorithms/quick-sort/

[3] https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

[4] http://bigocheatsheet.com

The resulting sorted LocationExperience array is used to populate a ListView[5] as shown in *Figure 4*.

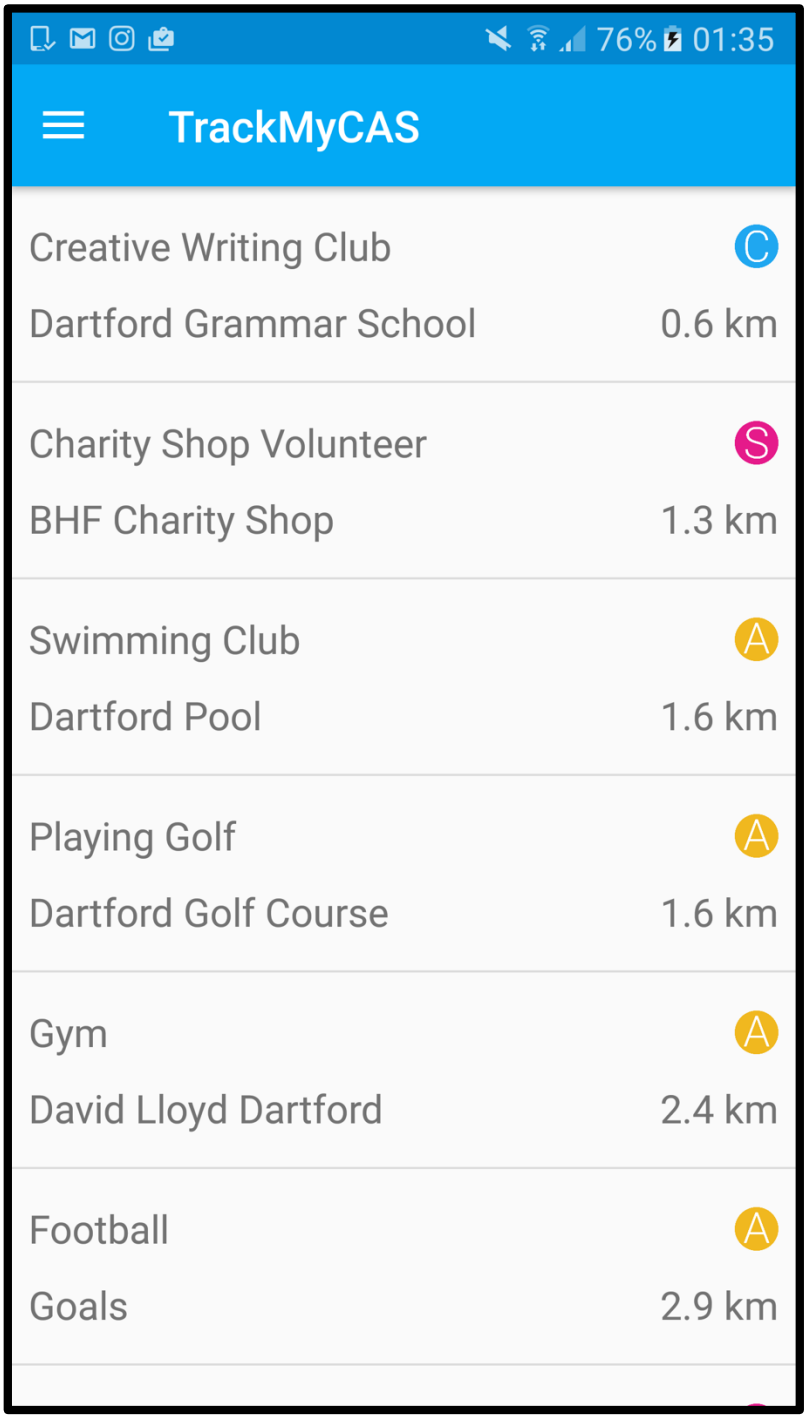

Figure 4

[5] https://developer.android.com/reference/android/widget/ListView.html

3. Implementing Parcelable Interface

The LocationExperience class implements the Parcelable[6] interface, which is built into the Android API. Therefore, it has the methods that are shown in *Figure 5*.

```
/* everything below here is for implementing Parcelable */
/* PARCELABLE ALLOWS FOR EXPERIENCE OBJECTS TO BE PASSED TO LOCATIONDETAILACTIVITY IN THE INTENT!! */

@Override
public int describeContents() {
    return 0;
}

// write  object's data to the passed-in Parcel
@Override
public void writeToParcel(Parcel out, int flags) {
    super.writeToParcel(out, flags);
    out.writeDouble(dist);
    out.writeString(locationid);
    out.writeString(locationname);
    out.writeDouble(latitude);
    out.writeDouble(longitude);
    out.writeLong(experienceid);
}

// this is used to regenerate  object. All Parcelables must have a CREATOR that implements these two methods
public static final Parcelable.Creator<Experience> CREATOR = new Parcelable.Creator<Experience>() {
    public LocationExperience createFromParcel(Parcel in) { return new LocationExperience(in); }

    public LocationExperience[] newArray(int size) {
        return new LocationExperience[size];
    }
};

protected LocationExperience(Parcel in) {
    super(in);
    dist = in.readDouble();
    locationid = in.readString();
    locationname = in.readString();
    latitude = in.readDouble();
    longitude = in.readDouble();
    experienceid = in.readInt();
}
```

Figure 5

These methods, which includes a constructor which takes a Parcel as a parameter, allow for LocationExperience objects to be passed between Activities in the intents. This is utilised when a user taps one of the ListView items in the NearMe Activity, whereby the LocationExperience object that relates to the selected item is passed to the LocationDetail Activity allowing its data to be displayed. This is **evidence of ingenuity** since it means that the data for the locations only has to be requested from the server once, and can then be accessed from different parts of the app, which means that loading times are reduced.

[6] https://developer.android.com/reference/android/os/Parcelable.html

4. Image Serialization

Images were serialized[7] and then encoded as Base64[8] in order to transfer them to the server, where they are then deserialized and saved in the filesystem. This is an example of **ingenuity** because it means that the images can be transmitted through the Hyper-Text Transfer Protocol[9] (which only works with Strings), similar to other data, and as such means that there is less overhead.

```
bitmap = bitmap.createScaledBitmap(bitmap,width,height,false);

ByteArrayOutputStream baos = new ByteArrayOutputStream();
bitmap.compress(Bitmap.CompressFormat.JPEG, 80, baos);
byte[] imageBytes = baos.toByteArray();
final String encodedImage = Base64.encodeToString(imageBytes, Base64.DEFAULT);

Connections.UploadImageConnection runner = new Connections.UploadImageConnection();
runner.setContext(AddNewExperienceActivity.this);
runner.execute(encodedImage);
```

Figure 6

Figure 6 shows how the Bitmap image is first compressed in the JPEG[10] format **which is clever** because it will minimise file size and therefore transmission time, before being converted to a byte array – it's serialized form. The bytes are then encoded as a String using Base64 encoding which can then be sent over HTTP on the network thread.

[7] https://en.wikipedia.org/wiki/Serialization
[8] http://stackoverflow.com/questions/3538021/why-do-we-use-base64
[9] https://www.w3.org/Protocols/rfc2616/rfc2616.html
[10] https://jpeg.org/jpeg/

5. PHP Script to Make SQL Queries

```php
<?php
    $dbservername = "localhost";
    $dbusername = "trackmycas";
    $dbpassword = "dartfordgs";
    $dbname = "trackmycas";

    // Create connection
    $conn = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
    // Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }

    $email = $_POST["email"];
    $password = $_POST["password"];
    $sql = "SELECT id,firstname,hash FROM users WHERE email = '$email'";
    $result = mysqli_query($conn, $sql);

    $loggedin = false;
    $userid = 0;
    if (mysqli_num_rows($result) > 0) {
        // output data of each row
        while($row = mysqli_fetch_assoc($result)) {
            if (password_verify($password,$row["hash"])) {
                $loggedin = true;
                $userid = $row["id"];
            } else {
                echo "Authentification Error";
            }
        }
    } else {
        echo "Authentification Error";
    }

    if ($loggedin) {
        $sql = "SELECT * FROM experiences WHERE userid = $userid";
        $result = mysqli_query($conn, $sql);
        $rows = array();
        if (mysqli_num_rows($result) > 0) {
            // output data of each row
            while($row = mysqli_fetch_assoc($result)) {
                $rows[] = $row;
            }
        } else {
            echo "No Records Found.";
        }

        echo json_encode($rows);
    }

    mysqli_close($conn);
?>
```

Figure 7

Figure 7 shows one of the PHP scripts that the app posts the data to. Firstly, a database connection is set up. Then the data from the app is collected from the $_POST superglobal[11]. The script then queries the database, returning the record with the passed email. If the passed password matches, for security, the database entry, then the script continues. The script then makes another SQL[12] query, selecting all columns from all the rows where the userid matches the userid of the logged in user. The resulting data is stored in an array, with a row in each index. This array is then encoded as JSON[13] to be read by the BufferedReader similar to as shown in *Figure 1*. Placing a PHP script between the database and the application **shows ingenuity** because it means that in future, versions for different operating systems can be built (e.g. iOS) without having to recode the server-side database logic.

[11] http://php.net/manual/en/reserved.variables.post.php
[12] http://www.w3schools.com/sql/sql_intro.asp
[13] http://www.json.org

6. JSON & Gson Java Library

In order to send data from the server back to the application, the JSON (JavaScript Object Notation)[14] format was used. This **is ingenious** because it means that Java objects can be created directly from the data that is returned from the server following the SQL query. An example of the JSON returned by the server is shown in the terminal printout in *Figure 8*.

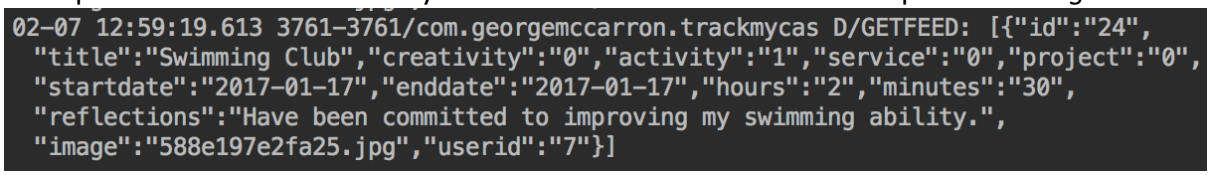

Figure 8

After the JSON has been received, the a new Gson[15] Object is created, which is then used to populate a Java array of LocationExperience objects, based on the objects contained within the JSON array, as shown in *Figure 9*. The use of an external library meant that this could be achieved without writing extensive code, thus speeding up development time, to meet the schedule agreed with the client.

```
Gson g = new Gson();
experiences = g.fromJson(result, Experience[].class);
```

Figure 9

[14] http://www.w3schools.com/js/js_json_syntax.asp
[15] https://github.com/google/gson

7. Writing to PDF File

In order to build the Export feature, it was necessary to write data to a PDF and save the file to the phone's secondary storage. This was achieved using the iTextPDF[16] external library, which **shows ingenuity** because it means that less code had to be written, saving on development time. The library was added as a Gradle[17] dependency as shown in *Figure 10*.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.0.0'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:design:25.0.0'
    compile 'com.google.code.gson:gson:2.7'
    compile 'com.itextpdf:itextg:5.5.10'
    compile 'com.google.android.gms:play-services:10.0.1'
}
```

Figure 10

Firstly, the file is created using the code showing in *Figure 11*. In order to ensure that each file has a unique name, a timestamp is generated, and is used for the filename.

```
File pdfFolder = new File(Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_DOCUMENTS), "TrackMyCASReport");
if (!pdfFolder.exists()) {
    pdfFolder.mkdir();
    Log.i("PDFLOG", "Pdf Directory created");
}

//Create time stamp
Date date = new Date() ;
String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(date);

File myFile = new File(pdfFolder, user.getFirstName() + user.getLastName() + timeStamp + ".pdf");

OutputStream output = null;
try {
    output = new FileOutputStream(myFile);
} catch (FileNotFoundException e) {
    e.printStackTrace();
}

//Step 1
Document document = new Document();

//Step 2
try {
    PdfWriter.getInstance(document, output);
} catch (DocumentException e) {
    e.printStackTrace();
}

//Step 3
document.open();

// Set the fonts
```

Figure 11

Following the creation of the empty file, it is populated by looping through each element of an array of Experience objects, and accessing the fields of the object. The document.add() method[18] is used to write strings to the file, as shown in *Figure 12*.

---

[16] http://itextpdf.com

[17] https://docs.gradle.org/current/userguide/userguide.html

[18] http://developers.itextpdf.com/examples/itext-action-second-edition/chapter-1

```
for (int i = 0; i < experiences.length; i++) {
    try {
        Paragraph all = new Paragraph();
        if (experiences[i].getStartdate().equals(experiences[i].getEnddate())) {
            String[] dateParts = experiences[i].getStartdate().split("-");
            all.add(new Chunk(dateParts[2] + "/" + dateParts[1] + "/" + dateParts[0] + ": ",dateFont));
        } else {
            String[] sdateParts = experiences[i].getStartdate().split("-");
            String[] edateParts = experiences[i].getEnddate().split("-");
            all.add(new Chunk(sdateParts[2] + "/" + sdateParts[1] + "/" + sdateParts[0] + " - " +
                    edateParts[2] + "/" + edateParts[1] + "/" + edateParts[0] + ": ",dateFont));
        }
        if (experiences[i].isProject()) {
            all.add(new Chunk("CAS Project - ",projectFont));
        }
        all.add(new Chunk(experiences[i].getTitle(),h1Font));
        document.add(all);
    } catch (DocumentException e) {
        e.printStackTrace();
    }
    String strandInfo = "";
    if(experiences[i].isCreativity()){
        strandInfo = strandInfo + "Creativity";
        if (experiences[i].isActivity()) {
            strandInfo = strandInfo + ", Activity";
            if (experiences[i].isService()) {
                strandInfo = strandInfo + ", Service";
            }
        }
        else if (experiences[i].isService()) {
            strandInfo = strandInfo + ", Service";
        }
    }
    else if(experiences[i].isActivity()) {
        strandInfo = strandInfo + "Activity";
        if (experiences[i].isService()) {
            strandInfo = strandInfo + ", Service";
        }
    }
    else if(experiences[i].isService()){
        strandInfo = strandInfo + "Service";
    }
```

Figure 12